Package 'gsubfn'

October 13, 2022

Version 0.7

Date 2018-03-15

Title Utilities for Strings and Function Arguments

Author G. Grothendieck

Maintainer G. Grothendieck <ggrothendieck@gmail.com>

Description The gsubfn function is like gsub but can take a replacement
function or certain other objects instead of the replacement string.
Matches and back references are input to the replacement function and
replaced by the function output. gsubfn can be used to split strings
based on content rather than delimiters and for quasi-perl-style string
interpolation. The package also has facilities for translating formulas
to functions and allowing such formulas in function calls instead of
functions. This can be used with R functions such as apply, sapply,
lapply, optim, integrate, xyplot, Filter and any other function that
expects another function as an input argument or functions like cat
or sql calls that may involve strings where substitution is desirable.
There is also a facility for returning multiple objects from functions
and a version of transform that allows the RHS to refer to LHS used in
the same transform.

BugReports https://github.com/ggrothendieck/gsubfn/issues

Depends proto

Suggests boot, chron, grid, lattice, quantreg, svUnit, tcltk, zoo

License GPL (>= 2)

URL https://github.com/ggrothendieck/gsubfn

NeedsCompilation no

Repository CRAN

Date/Publication 2018-03-16 08:58:03 UTC

R topics documented:

---

gsubfn-package   *gsubfn*

Description

Generalized "'gsub'" and associated functions.

Details

gsubfn is an R package used for string matching, substitution and parsing. A seemingly small generalization of gsub, namely allow the replacement string to be a replacement function, list, formula or proto object, can result in significantly increased power and applicability. The resulting function, gsubfn is the namesake of this package. In the case of a replacement formula the formula is interpreted as a function with the body of the function represented by the right hand side of the formula. In the case of a replacement proto object the object space is used to store persistant data to be communicated from one function invocation to the next as well as to store the replacement function/method itself.

Built on top of gsubfn is strapply which is similar to gsubfn except that it returns the output of the function rather than substituting it back into the source string.

A fast version of strapply specialized to the function c is provided.

The ability to have formula arguments that represent functions can be used not only in the functions of the gsubfn package but can also be used with any R function that itself passes functions without modifying its source. Such functions might include apply, lapply, mapply, sapply, tapply, by, integrate, optim, outer and other functions in the core of R and in addon packages. Just preface any R function with fn\$ and subject to certain rules which are intended to distinguish which formulas are intended to be functions and which are not, the formula arguments will be translated to functions, e.g. fn$integrate(~ x^2, 0, 1) fn\$ also performs quasi-perl style string interpolation on any character arguments beginning with \1.

match.funfn, is provided to allow developers to readily build this functionality into their own functions so that even the fn\$ prefix need not be used.

The home page plus the following are sources of information on "gsubfn":

Home Page (see URL: line)	RShowDoc("DESCRIPTION", package = "gsubfn")
News	RShowDoc("NEWS", package = "gsubfn")
Wish List	RShowDoc("WISHLIST", package = "gsubfn")

Thanks file	RShowDoc("THANKS", package = "gsubfn")
License	RShowDoc("COPYING", package = "gsubfn")
Citation	citation(package = "gsubfn")
Demo	demo("gsubfn-chron")
Demo	demo("gsubfn-cut")
Demo	demo("gsubfn-gries")
Demo	demo("gsubfn-si")
Unit tests	demo("gsubfn-unitTests")
This File	package?gsubfn
Help files	?gsubfn, ?strapply, ?cat0
More Help files	?as.function.formula, ?match.funfn, ?fn
Home page	http://code.google.com/p/gsubfn/
Vignette	vignette("gsubfn")

Examples

```
# replace each number with that number plus 1
gsubfn("[[:digit:]]+", function(x) as.numeric(x)+1, "(10 20)(100 30)")

# same
gsubfn("[[:digit:]]+", ~ as.numeric(x)+1, "(10 20)(100 30)")

# replace each number with its cumulative sum
pcumsum <- proto(pre = function(this) this$sum <- 0,
   fun = function(this, x) { sum <<- sum + as.numeric(x) }
)
gsubfn("[0-9]+", pcumsum, "10 abc 5 1")

# split out numbers
strapply("12abc34 55", "[0-9]+")

fn$optim(1, ~ x^2, method = "CG")

fn$integrate(~ sin(x) + cos(x), 0, pi/2)

fn$lapply(list(1:4, 1:5), ~ LETTERS[x]) # list(LETTERS[1:4], LETTERS[1:5])

fn$mapply(~ seq_len(x) + y * z, 1:3, 4:6, 2) # list(9, 11:12, 13:15)

# must specify x since . is a free variable
fn$by(CO2[4:5], CO2[1], x ~ coef(lm(uptake ~ ., x)), simplify = rbind)

# evaluate f at x^2 where f may be function or formula
square <- function(f, x, ...) { f <- match.funfn(f); f(x^2, ...) }
square(~ exp(x)/x, pi)
square(function(x) exp(x)/x, pi) # same
```

---

as.function.formula   *Make a one-line function from a formula.*

Description

Create a function from a formula.

Usage

```
## S3 method for class 'formula'
as.function(x, ...)
```

Arguments

x   Formula with no left side.

...   Currently not used.

Value

A function is returned whose formal arguments are the variables in the left hand side, whose body
is the expression on the right side of the formula and whose environment is the environment of the
formula. If there is no left hand side the free variables on the right, in the order encountered are
used as the arguments. letters, LETTERS and pi are ignored and not used as arguments. If there
is no left hand side and any of ..1, ..2, ..., ..9 are found as free variables then they are not used
as arguments but ... is used as a trailing argument instead. If there is no left hand side and `&` is
found as a free variable then that variable is used as the first argument and ... is added as the last
argument. If the left hand side is 0 then the function is created as a zero argument function.

Note

->, ->>, =, <-, <<- and ? all have lower operator precdence than ~ so function bodies that contain
them typically must be surrounded with {...}.

See Also

Syntax.

Examples

```
old.options <- options(keep.source = FALSE)
as.function(~ as.numeric(x) + as.numeric(y))
as.function(x + y ~ as.numeric(x) + as.numeric(y)) # same
as.function(~ ..1 + ..2)
# the replacement function in gsubfn uses as.function.formula to
# interpret formulas as functions.  Here we insert ! after each digit.
gsubfn("[0-9]", ~ paste0(`&`, "!"), "ab4cd5")
## Not run:
# example where function body must be surrounded with {...}
```

```
# due to use of <<-.  See warning section above.
assign("mywarn", NULL, .GlobalEnv)
fn$tryCatch( warning("a warning"),
    warning = w ~ { mywarn <<- conditionMessage(w)})
print(mywarn)

## End(Not run)
options(old.options)
```

---

fn   *Transform formula arguments to functions.*

Description

When used in the form fn\$somefunction(...arguments...) it converts formulas among the arguments of somefunction to functions using `as.function.formula`. It uses a heuristic to decide which formulas to convert. If any of the following are true then that argument is converted from a formula to a function: (1) there is only one formula among the arguments, (2) the name of the formula argument is FUN or (3) the formula argument is not the first argument in the argument list.

It also removes any `simplify` argument whose value is not logical and after processing it in the same way just discussed in order to interpret it as a function it passes the output of the command through do.call(simplify, output).

It also performs quasi-perl style string interpolation on any character string arguments that begin with \1 removing the \1 character. A dollar sign followed by a variable name or R code within backticks are both evaluated.

Usage

```
## S3 method for class 'fn'
x$FUN
```

Arguments

x   fn.

FUN   Name of a function.

Value

Returns a function.

See Also

as.function.formula

Examples

```
# use of formula to specify a function.
# Note that LETTERS, letters and pi are automatically excluded from args
fn$lapply(list(1:4, 1:3), ~ LETTERS[x])
fn$sapply(1:3, ~ sin((n-1) * pi/180))

# use of simplify = rbind instead of do.call(rbind, by(...)).
# args to anonymous function are automatically determined.
fn$by(BOD, 1:nrow(BOD), ~ c(mn = min(x), mx = max(x)), simplify = rbind)

# calculate lm coefs of uptake vs conc for each Plant
fn$by(CO2, CO2$Plant, d ~ coef(lm(uptake ~ conc, d)), simplify = rbind)

# mid range of conc and uptake by Plant
fn$aggregate(CO2[,4:5], CO2[1], ~ mean(range(x)))

# string interpolation
j <- fn$cat("pi = $pi, exp = `exp(1)`\n")

## Not run:

# same but use cast/melt from reshape package
library(reshape)
fn$cast(Plant ~ variable, data = melt(CO2, id = 1:3), ~~ mean(range(x)))

# same
# uncomment when new version of doBy comes out (expected shortly)
# library(doBy)
# fn$summaryBy(.~Plant,CO2[-(2:3)],FUN= ~~mean(range(x)), pref='midrange')

## End(Not run)

# generalized matrix product
# can replace sum(x*y) with any other inner product of interest
# this example just performs matrix multiplication of a times b
a <- matrix(4:1, 2)
b <- matrix(1:4, 2)
fn$apply(b, 2, x ~ fn$apply(a, 1, y ~ sum(x*y)))

# integration
fn$integrate(~1/((x+1)*sqrt(x)), lower = 0, upper = Inf)

# optimization
fn$optimize(~ x^2, c(-1,1))

# using fn with S4 definitions
setClass('ooc', representation(a = 'numeric'))
fn$setGeneric('incr', x + value ~ standardGeneric('incr'))
fn$setMethod('incr', 'ooc', x + value ~ {x@a <- x@a+value; x})
oo <- new('ooc',a=1)
oo <- incr(oo,1)
```

```
    oo

## Not run:

    # plot quantile regression fits for various values of tau
    library(quantreg)
    data(engel)
    plot(engel$x, engel$y, xlab = 'income', ylab = 'food expenditure')
    junk <- fn$lapply(1:9/10, tau ~ abline(coef(rq(y ~ x, tau, engel))))

    # rolling mid-range
    library(zoo)
    fn$rollapply(LakeHuron, 12, ~ mean(range(x)))

    library(lattice)
    fn$xyplot(uptake ~ conc | Plant, CO2,
       panel = ... ~ { panel.xyplot(...); panel.text(200, 40, lab = 'X') })

    library(boot)
    set.seed(1)
    fn$boot(rivers, ~ median(x, d), R = 2000)

## End(Not run)

    x <- 0:50/50
    matplot(x, fn$outer(x, 1:8, ~ sin(x * k*pi)), type = 'blobcsSh')
```

---

gsubfn   *Pattern Matching and Replacement*

Description

Like gsub except instead of a replacement string one uses a function which accepts the matched text as input and emits replacement text for it.

Usage

```
gsubfn(pattern, replacement, x, backref, USE.NAMES = FALSE,
ignore.case = FALSE, engine = getOption("gsubfn.engine"),
env = parent.frame(), ...)
```

Arguments

pattern	Same as pattern in gsub
replacement	A character string, function, list, formula or proto object. See Details.
x	Same as x in gsub

backref   Number of backreferences to be passed to function. If zero or positive the match
                 is passed as the first argument to the replacement function followed by the in-
                 dicated number of backreferences as subsequent arguments. If negative then
                 only the that number of backreferences are passed but the match itself is not. If
                 omitted it will be determined automatically, i.e. it will be 0 if there are no back-
                 references and otherwise it will equal negative the number of back references.
                 It determines this by counting the number of non-escaped left parentheses in
                 the pattern. Also if the function contains an ampersand as an argument then
                 backref will be taken as non-negative and the ampersand argument will get the
                 full match.

USE.NAMES   See USE.NAMES in sapply.

ignore.case   If TRUE then case is ignored in the pattern argument.

engine   Specifies which engine to use. If the R installation has tcltk capability then the
                 tcl engine is used unless FUN is a proto object or perl=TRUE in which case the
                 "R" engine is used (regardless of the setting of this argument).

env   Environment in which to evaluate the replacement function. Normally this is
                 left at its default value.

...   Other gsub arguments.

Details

If replacement is a string then it acts like gsub.

If replacement is a function then each matched string is passed to the replacement function and
the output of that function replaces the matched string in the result. The first argument to the
replacement function is the matched string and subsequent arguments are the backreferences, if
any.

If replacement is a list then the result of the regular expression match is, in turn, matched against
the names of that list and the value corresponding to the first name in the list that is match is
returned. If there are no names matching then the first unnamed component is returned and if there
are no matches then the string to be matched is returned. If backref is not specified or is specified
and is positive then the entire match is used to lookup the value in the list whereas if backref is
negative then the identified backreference is used.

If replacement is a formula instead of a function then a one line function is created whose body is
the right hand side of the formula and whose arguments are the left hand side separated by + signs
(or any other valid operator). The environment of the function is the environment of the formula.
If the arguments are omitted then the free variables found on the right hand side are used in the
order encountered. 0 can be used to indicate no arguments. letters, LETTERS and pi are never
automatically used as arguments.

If replacement is a proto object then it should have a fun method which is like the replacement
function except its first argument is the object and the remaining arguments are as in the replacement
function and are affected by backref in the same way. gsubfn automatically inserts the named
arguments in the call to gsubfn into the proto object and also maintains a count variable which
counts matches within strings. The user may optionally specify pre and post methods in the proto
object which are fired at the beginning and end of each string (not each match). They each take one
argument, the object.

Note that if the "R" engine is used and if backref is non-negative then internally the pattern will be parenthesized.

A utility function cat0 is available. They are like [cat](#) and [paste](#) except that their default sep value is "".

Value

As in gsub.

See Also

[strapply](#)

Examples

```
# adds 1 to each number in third arg
gsubfn("[[:digit:]]+", function(x) as.numeric(x)+1, "(10 20)(100 30)")

# same but using formula notation for function
gsubfn("[[:digit:]]+", ~ as.numeric(x)+1, "(10 20)(100 30)")

# replaces pairs m:n with their sum
s <- "abc 10:20 def 30:40 50"
gsubfn("([0-9]+):([0-9]+)", ~ as.numeric(x) + as.numeric(y), s)

# default pattern for gsubfn does quasi-perl-style string interpolation
gsubfn( , , "pi = $pi, 2pi = `2*pi`")

# Extracts numbers from string and places them into numeric vector v.
# Normally this would be done in strapply instead.
v <- c(); f <- function(x) v <<- append(v,as.numeric(x))
junk <- gsubfn("[0-9]+", f, "12;34:56,89,,12")
v

# same
strapply("12;34:56,89,,12", "[0-9]+", simplify = c)

# replaces numbers with that many Xs separated by -
gsubfn("[[:digit:]]+", ~ paste(rep("X", n), collapse = "-"), "5.2")

# replaces units with scale factor
gsubfn(".m", list(cm = "e1", km = "e6"), "33cm 45km")

# place <...> around first two occurrences
p <- proto(fun = function(this, x) if (count <= 2) paste0("<", x, ">") else x)
gsubfn("\\w+", p, "the cat in the hat is back")

# replace each number by cumulative sum to that point
p2 <- proto(pre = function(this) this$value <- 0,
fun = function(this, x) this$value <- value + as.numeric(x))
gsubfn("[0-9]+", p2, "12 3 11, 25 9")
```

```
# this only works if your R installation has tcltk capabilities
# See following example for corresponding code with R engine
if (isTRUE(capabilities()[["tcltk"]])) {
gsubfn("(.)\\1", ~ paste0(`&`, "!"), "abbcddd")
}

# with R and backref >=0 (implied) the pattern is internally parenthesized
# so must use \2 rather than \1
gsubfn("(.)\\2", ~ paste0(`&`, "!"), "abbcddd", engine = "R")
```

---

list   *Multiple value assignment.*

Description

Multiple value assignment.

Details

Enables list to be used on the left hand side of assignment statements as shown in the examples.

Examples

```
# swap a and b without explicitly creating a temporary
a <- 1; b <- 2
list[a,b] <- list(b,a)

# get eigenvectors and eigenvalues
list[eval, evec] <- eigen(cbind(1,1:3,3:1))

# get today's month, day, year
require(chron)
list[Month, Day, Year] <- month.day.year(unclass(Sys.Date()))

# get first two components of linear model ignoring rest
list[Coef, Resid] <- lm(rnorm(10) ~ seq(10))

# assign Green and Blue (but not Red) components
list[,Green,Blue]  <- col2rgb("aquamarine")

# Assign QR and QRaux but not other components
list[QR,,QRaux]   <- qr(c(1,1:3,3:1))
```

---

match.funfn *Generic extended version of R match.fun*

Description

A generic `match.fun`.

Usage

```
match.funfn(FUN, descend = TRUE)
```

Arguments

FUN   Function, character name of function or formula describing function.

descend   logical; control whether to search past non-function objects.

Details

The default method is the same as `match.fun` and the `formula` method is the same as `as.function.formula`. This function can be used within the body of a function to convert a function specification whether its a function, character string or formula into an actual function.

Value

Returns a function.

See Also

See Also `match.fun`, `as.function.formula`.

Examples

```
# return first argument evaluated at second argument squared.
sq <- function(f, x) {
   f <- match.funfn(f)
   f(x^2)
}

# call sq using different forms for function
sq(function(x) exp(x)/x, pi)
f <- function(x) exp(x)/x
sq("f", pi)
sq(~ exp(x)/x, pi)
sq(x ~ exp(x)/x, pi)
```

---

read.pattern   *Read file or text string using a regular expression to separate fields.*

Description

Read file or text string using a regular expression to separate fields.

Usage

```
read.pattern(file, pattern, perl = FALSE, text, sep = "\01", fileEncoding = "", ...)
```

Arguments

file	See `read.table`.
pattern	A regular expression.
perl	logical. If TRUE then a perl style regular expression is expected.
text	See `read.table`.
sep	This should be a character that does not appear in any field. It is used for holding an intermediate form of the text.
fileEncoding	See `read.table`.
...	Further arguments passed to `read.table`.

Details

The parenthesized portions of the regular expression are extracted as fields.

Value

A data.frame.

See Also

See `strapplyc`, `read.table`.

Examples

```
# 3 space-separated fields.  The first field may have embedded spaces.
Lines <- "this is the first field 1 2
more text 3 4
"
pat <- "^(.*) +(\\S+) +(\\S+)$"
read.pattern(text = Lines, pattern = pat, as.is = TRUE)

# 3 space-separated fields.  The last field may have embedded spaces.
Lines2 <- "1 2 this is the first field
3 4 more text
"
```

```
pat2 <- "^(\\S+) +(\\S+) +(.*)$"
read.pattern(text = Lines2, pattern = pat2, as.is = TRUE)
```

---

strapply   *Apply a function over a string or strings.*

Description

Similar to "'gsubfn'" except instead of performing substitutions it returns the output of "'FUN'".

Usage

```
strapply(X, pattern, FUN = function(x, ...) x, backref, ..., empty,
ignore.case = FALSE, perl = FALSE, engine,
simplify = FALSE, USE.NAMES, combine = c)
strapplyc(X, pattern, backref, ignore.case = FALSE, simplify = FALSE, USE.NAMES, engine)
```

Arguments

X	list or (atomic) vector of character strings to be used.
pattern	character string containing a regular expression (or character string for "'fixed = TRUE')" to be matched in the given character vector.
FUN	a function, formula, character string, list or proto object to be applied to each element of "'X'". See discussion in [gsubfn]().
backref	See [gsubfn]().
empty	If there is no match to a string return this value.
ignore.case	If TRUE then case is ignored in the pattern argument.
perl	If TRUE then engine="R" is used with perl regular expressions.
engine	This argument defaults to getOption("gsubfn.engine"). If that option has not been set engine defaults to the "R" engine if (1) FUN is a proto object or if (2) the R installation does not have tcltk capability. If the "R" default does not apply then it defaults to the "tcl" engine.
...	optional arguments to "'gsubfn'".
simplify	logical or function. If logical, should the result be simplified to a vector or matrix, as in "sapply" if possible? If function, that function is applied to the result with each component of the result passed as a separate argument. Typically if the form is used it will typically be specified as rbind.
USE.NAMES	logical; if "'TRUE'" and if "'X'" is character, use "'X'" as 'names' for the result unless it had names already. Default is FALSE.
combine	combine is a function applied to the components of the result of FUN. The default is "c". "list" is another common choice. The default may change to be "list" in the future.

Details

If FUN is a function then for each character string in "X" the pattern is repeatedly matched, each such match along with back references, if any, are passed to the function "FUN" and the output of FUN is returned as a list. If FUN is a formula or proto object then it is interpreted to the way discussed in [gsubfn](#).

If FUN is a proto object or if perl=TRUE is specified then engine="R" is used and the engine argument is ignored.

If backref is not specified and engine="R" is specified or implied then a heuristic is used to calculate the number of backreferences. The primary situation that can fool it is if there are parentheses in the string that are not back references. In those cases the user will have to specify backref. If engine="tcl" then an exact algorithm is used and the problem sentence never occurs.

strapplyc is like strapply but specialized to FUN=c for speed. If the "tcl" engine is not available then it calls strapply and there will be no speed advantage.

Value

A list of character strings.

See Also

See [gsubfn](#). For regular expression syntax used in tcl see http://www.tcl.tk/man/tcl8.6/TclCmd/re_syntax.htm and for regular expression syntax used in R see the help page for regex.

Examples

```
strapply("12;34:56,89,,12", "[0-9]+")

# separate leading digits from rest of string
# creating a 2 column matrix: digits, rest
s <- c("123abc", "12cd34", "1e23")
t(strapply(s, "^([[:digit:]]+)(.*)", c, simplify = TRUE))

# same but create matrix
strapply(s, "^([[:digit:]]+)(.*)", c, simplify = rbind)

# running window of 5 characters using 0-lookahead perl regexp
# Note that the three ( in the regexp will fool it into thinking there
# are three backreferences so specify backref explicitly.
x <- "abcdefghijkl"
strapply(x, "(.)(?=(....))",  paste0, backref = -2, perl = TRUE)[[1]]

# Note difference.  First gives character vector.  Second is the same.
# Third has same elements but is a list.
# Fourth gives list of two character vectors. Fifth is the same.
strapply("a:b c:d", "(.):(.)", c)[[1]]
strapply("a:b c:d", "(.):(.)", list, simplify = unlist) # same

strapply("a:b c:d", "(.):(.)", list)[[1]]
```

```
strapply("a:b c:d", "(.):(.)", c, combine = list)[[1]]
strapply("a:b c:d", "(.):(.)", c, combine = list, simplify = c) # same

# find second CPU_SPEED value given lines of config file
Lines <- c("DEVICE = 'PC'", "CPU_SPEED = '1999', '233'")
parms <- strapply(Lines, "[^ ',=]+", c, USE.NAMES = TRUE,
simplify = ~ lapply(list(...), "[", -1))
parms$CPU_SPEED[2]

# return first two words in each string
p <- proto(fun = function(this, x) if (count <=2) x)
strapply(c("the brown fox", "the eager beaver"), "\\w+", p)

## Not run:
# convert to chron
library(chron)
x <- c("01/15/2005 23:32:45", "02/27/2005 01:22:30")
x.chron <- strapply(x, "(../../....) (..:..:..)",  chron, simplify = c)

# time parsing of all 275,546 words from James Joyce's Ulysses
joyce <- readLines("http://www.gutenberg.org/files/4300/4300-8.txt")
joycec <- paste(joyce, collapse = " ")
system.time(s <- strapplyc(joycec, "\\w+")[[1]])
length(s) # 275546

## End(Not run)
```

---

transform2	*Like transform but allows right hand sides to refer to left hand sides.*

Description

In `transform` the right hand sides of the arguments may not refer to the left hand sides. `transform2` works like `transform` with the exception that a right hand side may refer to any of the variables on the left hand side of another argument. The argument defining the left hand side may come before or after the reference.

Usage

```
transform2(`_data`, ...)
```

Arguments

_data	A data frame.
...	Further arguments of the form tag=value.

Value

The modified value of _data.

See Also

[transform](), [within]()

Examples

```
## Not run:
# bad
transform(BOD, b = a, a = demand)

## End(Not run)
# OK.  a is set to demand and b is set to a so both wind up equal to demand
transform2(BOD, b = a, a = demand)
```

Index